OULU UNIVERSITY OF
APPLIED SCIENCES

Tamas Mucs

REMOTE CONTROLLED CAR WITH AN ANDROID PLATFORM

REMOTE CONTROLLED CAR WITH AN ANDROID PLATFORM

Tamas Mucs
Bachelor's Thesis
Spring 2016
Information Technology
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology

Author: Tamas Mucs
Title of the bachelor's thesis: Remote Controlled Car with an Android Platform
Supervisor: Lauri Pirttiaho
Term and year of completion: Autumn 2016 Number of pages: 47

The aim of this bachelor's thesis was to tweak an existing RC model car so that it can be controlled from an Android device with the help of a Raspberry Pi. The car should also have a camera attached which periodically sends images back to the user. The car should also measure some environment data, such as temperature and distance ahead the next physical obstacle. Finally, another aim was that the user should be able to control the movement of the car.

After the thesis topic had been accepted by the supervising teacher, some hardware was obtained. First the RC car was bought and then a controlling Raspberry Pi, too. It was not necessary to buy an Android device, because it was already owned before starting the thesis. After the gathering of the car and the Raspberry Pi, the implementation of the software began. Later on the final pieces of hardware were installed onto the circuit. Technologies covered in this task were embedded, and Android development.

In the end, the result of the work was a working prototype which meets the originally defined objectives. There are still some things that can be improved and perhaps new features could be also added. The purpose of this thesis though was not to have a completely perfect product. The purpose was rather to demonstrate and deepen the knowledge that is required to implement such a product.

Keywords: Android, telecommunication, Java, embedded

PREFACE

This bachelor's thesis was based on author's own idea and it was made without the physical presence of the supervising teacher. The contact, however, was kept between the author and the supervisor and feedback was constantly received by the supervisor. Communication happened by e-mailing.

The thesis was made with a little external help, mostly in the form of advice and suggestions.

Oulu, 20.10.2016
Tamas Mucs

TABLE OF CONTENTS

VOCABULARY

RPi	Raspberry Pi
RC	Remote Controlled
CPU	Central processing unit
OSI model	Open Systems Interconnection model
OS	Operating system
UML	Unified modeling language
R/W	Read/Write
OOP	Object oriented programming
OOD	Object oriented design
R/W	Read/Write

1 INTRODUCTION

The aim of this thesis was to tweak an existing toy-car so that it can be controlled from an Android tablet. This car (called a server from now on) also collects some information and sends it back to the tablet (called a client from now on). The concept was to use the existing knowledge with a little extension to create a prototype. It was not an objective to develop it to such an extent that it is ready to be released on the market. The aim was rather to gain experience through self-study as well as demonstrating the skills being used.

This topic was chosen because such a project more or less covers the skills of a to-be-graduated student in the field of information technology. In addition, the technologies that were used throughout the project were also fairly recent.

There were two main parts in this thesis. One part was to build an electric circuit which would operate the toy car itself. There is a brief introduction of most pieces of hardware that were used. There are also circuit diagrams that describe how the electronics are connected.

The other part was to develop a software which would control the hardware and execute certain operations that are requested by the user. Two pieces of software were developed altogether. The one was for the server. Which was programmed by using Oracle Java Standard Edition 8.

The other software was developed for the client. The client was implemented using the standard Android API level 23 (Marshmallow). The backend of the client software was originally developed with the standard Java language prior porting it onto Android platform. Some basic Linux knowledge was also necessary as the controlling platform of the hardware is a Raspberry PI (abbreviated as RPi from now on) which runs Linux (namely the distribution of Raspbian). Some parts of the project needed additional study to get the work done. A bigger emphasis of the work was on the software development part of the project.

1.1 The Raspberry PI platform

The central piece of hardware that controls all the others is a Raspberry Pi 2 Model B+. The original aim of the Raspberry Pi project (by the UK-based Raspberry Pi Foundation) was to create a small, credit card-sized computer to help teaching children the basics of computer science.

The module is well equipped with different kinds of input/output ports and it performs quite well in tasks that are related to this thesis project. The Raspberry Pi model 2 B+ has the following features:

- 4 USB ports
- 40 GPIO pins
- Full HDMI port
- Ethernet port
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core (1)

Hardware capabilities vary by models.

The Raspberry Pi Foundation developed a few Linux distributions for the Raspberry Pi. It was designed so that the end-user would use Python as the main scripting language, however several others, including C, C++, Java, Ruby, Perl, Smalltalk are also supported. (2)

1.2 Internet of Things

Electronic devices may be programmed to do automated tasks and collect data. Then they share this data within a network. This is called “Internet of things “.

The Internet of Things (also abbreviated as IoT) gives an opportunity to remotely operations and machines or supervise areas without the need of human interaction. This gives a basis eg. for modern medicine, smart homes, advanced environmental observations. Generally, it provides an excellent ground for scientific and other measurements.

1.3 Software development

Hardware on its own can hardly operate such a complex system as the one in this project. There are several ways how machine-code can be programmed onto a computer. In situations where a blazing fast performance is a must, the best way is to directly program the software with a so-called assembly language. Assembly code is then turned to machine code by a so-called assembler. This, however, results in such a situation where the source code is specific only to one certain CPU architecture. In addition, it would require much more time and effort to program such a software that was used in this thesis project. In fact, due to the complexity of the software, assembly would not be an option in our case. Assembly is also called a low-level programming language. (3)

Instead of deciding for an assembly language, a high-level programming language, namely Java was chosen. This had both ups and downs. On one hand, it resulted in a somewhat slower execution speed. A runtime environment was also needed to run the software (more on this later).

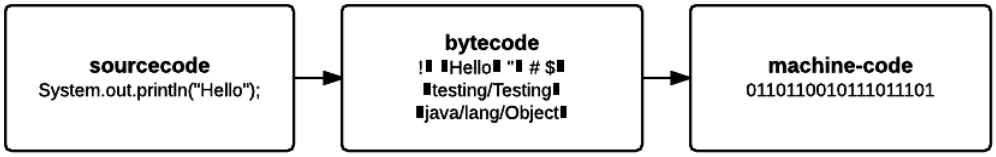

DIAGRAM 1. Compilation of a Java application

On the other hand, the source-code can be developed much faster, and it is also more extensible later on. In case of Java, the source code is first compiled into a so-called byte-code and upon the execution of the application this bytecode is turned into actual machine code by the Java Runtime Environment (JRE). This, of course, requires that the JRE is available for the particular system that will run the program. Thanks to this kind of architecture, one bytecode can be executed on any CPU architecture. This pattern is also called “write once, run anywhere” (WORA) (4).

2 THEORETICAL KNOWLEDGE PRIOR TO THE PROJECT

The topic for this thesis was chosen in such a way that the fundamental knowledge required for the project was more or less owned. Still, to completely accomplish this project, a little extension to the initial know-how was necessary.

2.1 Linux

It was necessary to have an operating system on the hardware that will be responsible for the execution of the software. In the case of this thesis project, Linux was a well-fitting choice. The RPi used in this project runs Raspbian, which is a Linux distribution built on top of Debian. The basic knowledge of Linux is useful when it comes to managing eg. R/W permissions, external package installations, basic configuration.

2.2 UML

In software development forward planning is needed. One way of doing it is to apply a visual designing methodology. UML (unified modeling language) provides a great visual overview of the architecture of the software. This helps not only the creators of the program to overview the system and develop more efficiently, but it also helps those who will later extend or maintain the existing code. Despite the positives of the UML, it is not frequently used in many projects. Many users consider it to beway too time consuming.

2.3 Object Oriented Programming

Building larger and more complex applications can become easier when writing in an object oriented language, rather than in a procedural language such, as C. In this way the code will also also be easier to extend.

The main principle about OOP is that the program code contains so called objects that store some data during the execution of the program. This data can be retrieved through functions (methods) and values (fields). Even though there are differences between object oriented programming languages, both syntax- and structure-wise, most of them are class-based. This means that objects in the code are instances of so-called classes. The type of the objects are determined by which classes they were instantiated (created) from. Generally, OOD boosts the extensibility, however a messy code structure might easily become unreadable and unreliable which might lead to undesired freezes and errors.

2.4 Development in a multi-threaded environment, handling concurrent data access issues.

Applications, which run several tasks at the same time, run each of their tasks (or processes) on their own so-called thread. However, this can bring up some issues that must be taken into account. Some programming languages provide native solutions to these problems.

Some problems may, eg. occur when at least two threads try to access a particular piece of data in the memory at the same time in such a way that at least one of them is doing a *write* operation. Then it is possible that data integrity gets broken, which means that unexpected values will be created and stored during runtime which can be quite hard to debug. It is advised that within the critical sections (sections of the code, such as variables and functions that are accessed simultaneously by several threads) it is regularly checked whether their value is valid and within an expected range. Another good idea is to draw a resource allocation graph which gives a visual overview on which process (thread) uses each resource. This also lessens the risk of deadlocks during the runtime. While there are several ways how to eliminate bugs caused by concurrent data access, it is a good practice to isolate unrelated information so that they are not stored in the same chunk of data.

2.5 Development using Oracle Java Standard Edition platform

On the raspberry PI platform, which is placed on the toy RC car, the Oracle Java platform was chosen as the programming language to do the necessary job. Here are the main reasons for selecting Java:

- Java is a platform independent, robust and powerful, garbage collected language. This means that unnecessary data is deleted automatically from the memory during the runtime. This lessens the required time for programming and the chance of memory leakage. Java provides a great support for multi-threading, too.
- Network communication can easily be implemented between the RPi and the controlling Android device as both of them support the networking socket of Java. Not to mention that a part of the code can be shared by the server (PRi) and the client (Android). This reduces time required for the implementation.

3 OVERVIEW OF THE HARDWARE

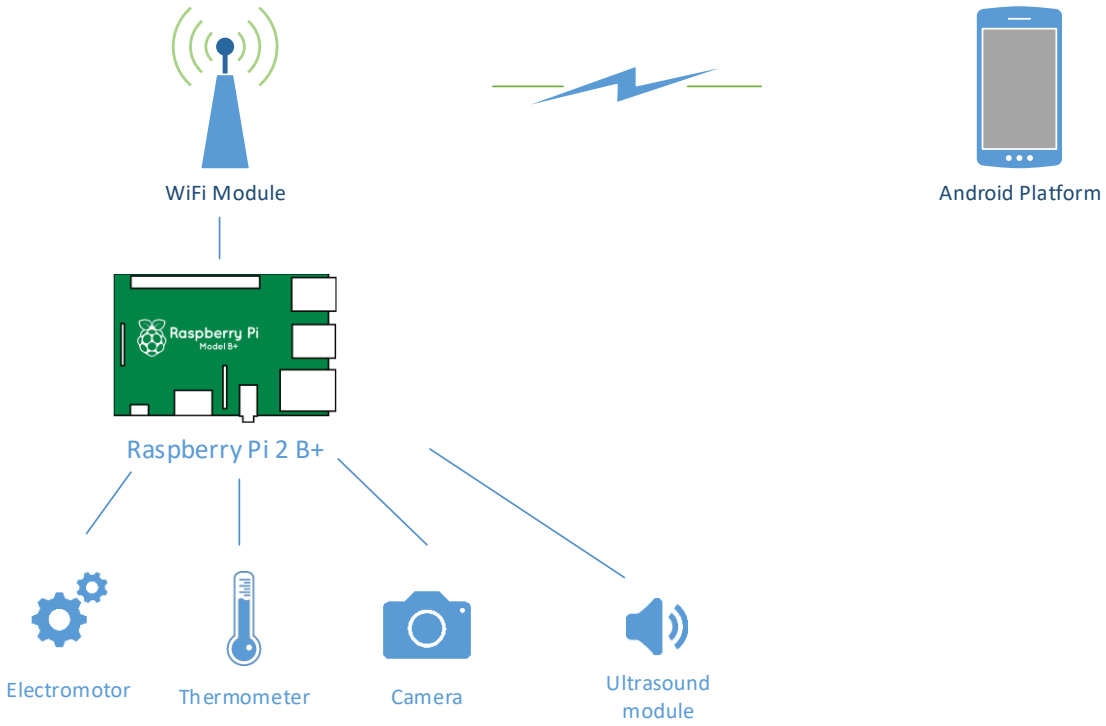

DIAGRAM 2. Overview of the system

There is a Raspberry Pi which controls the electromotors of the RC car. It reads temperature values from a thermometer, captures images with a camera module and measures the distance with an ultrasound module.

There is a connection established between the Raspberry Pi and the Android platform through a Wi-Fi module, which is connected to the USB port. The measured data is sent to the Android platform and movement commands are sent to the Raspberry Pi.

3.1 Necessary hardware

Table 1 shows which hardware is needed to implement the system that controls the RC car.

TABLE 1. Table of components

REF. NO.	TYPE/RATING	DESCRIPTION
C1		Raspberry Pi Camera
D1	1N4148	Diode
D2	1N4148	Diode
D3	1N4148	Diode
D4	1N4148	Diode
LED1		Green colored LED
LED2		Red colored LED
LED3		Green colored LED
LED4		Red colored LED
PB1	500 Basic	Adafruit BowerBoost 500 Basic
R1	270Ω	Resistor
R2	270Ω	Resistor
R3	270Ω	Resistor
R4	270Ω	Resistor
R5	4,7kΩ	Resistor
R6	330Ω	Resistor
R7	1kΩ	Resistor
REL1	G5V1	Relay

REL2	G5V1	Relay
REL3	G5V1	Relay
REL4	G5V1	Relay
T1	DS18B20+	1 Wire Temp-Sensor
U1	HC-SR04	Ultrasound Module

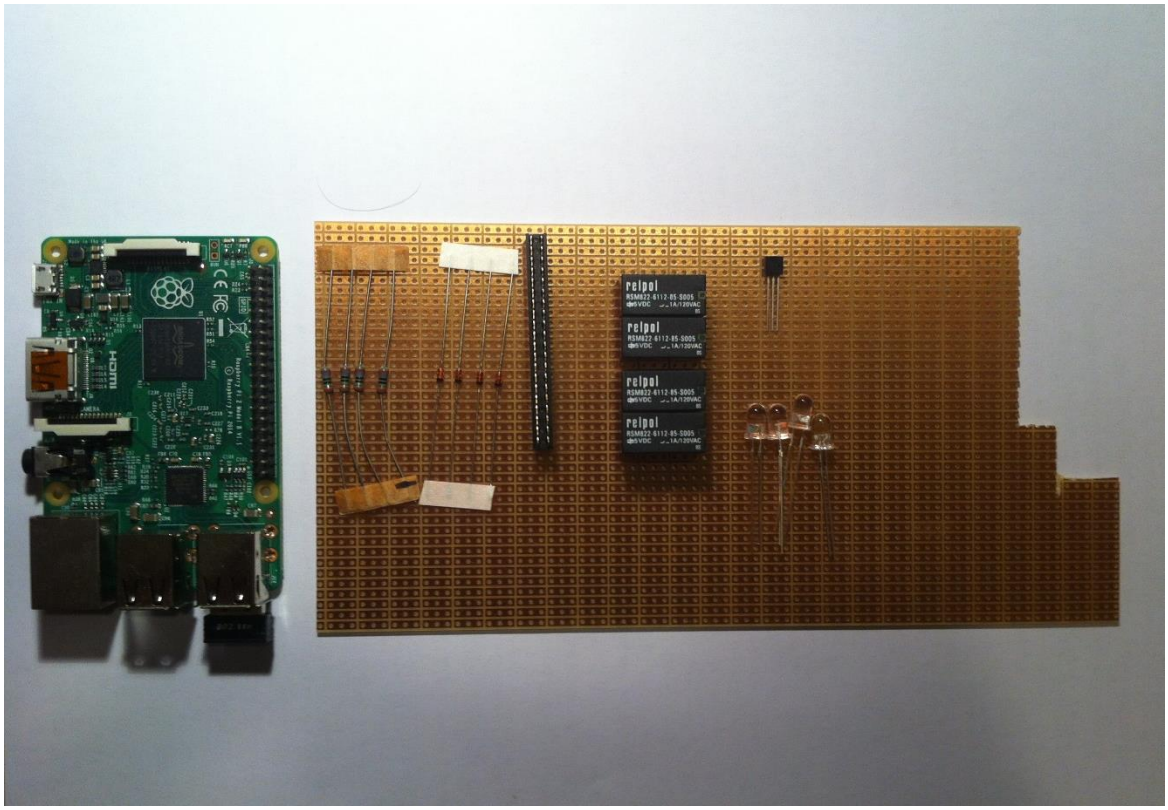

FIGURE 1. A non-complete image of the hardware needed.

Figure 1 above shows the initial components needed for the implementation. The relays labeled with *relpol* had to be replaced later on because they did not trigger upon the voltage provided by the Raspberry Pi.

3.2 The raspberry Pi

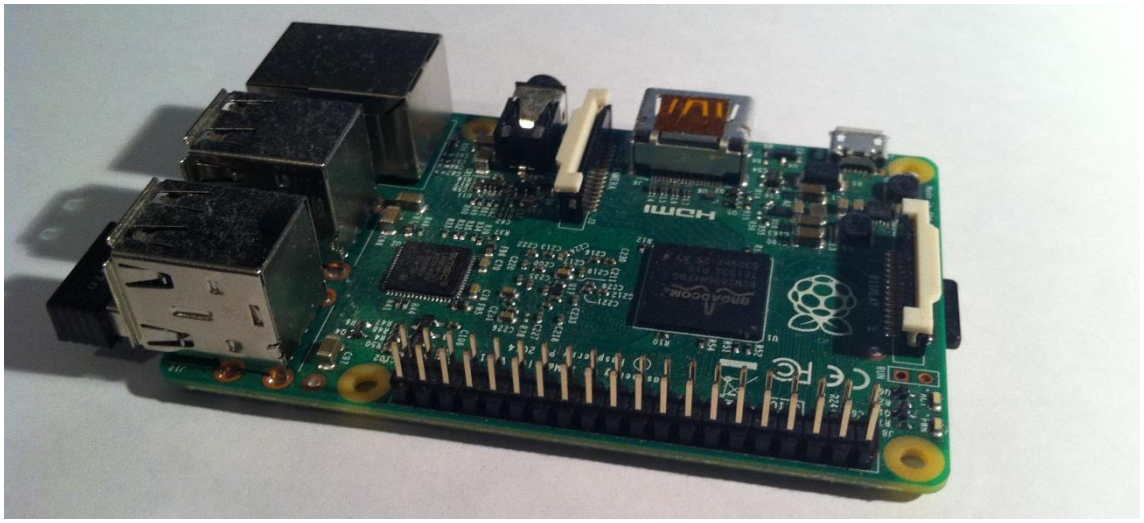

FIGURE 2. The Raspberry Pi platform

In the center of the hardware there is a Raspberry Pi (see figure 2). This controls the attached hardware, such as sensors and the camera.

The platform operates on 5 volts and has the following ports available:

- 5v input (For operation)
- 5x USB
- Ethernet
- 3.5mm audio jack
- CSI camera input
- DSI display output
- HDMI output
- 40 pin GPIO header
- micro SD slot

When powered on, the system boots from the micro SD card. There is a selection of system images available which can be flashed onto the bootable micro SD. After the system has been booted up, the controlling software launches. There is also a Wi-Fi module inserted to one of its USB ports to provide the connectivity to the LAN.

3.3 Panel with connector

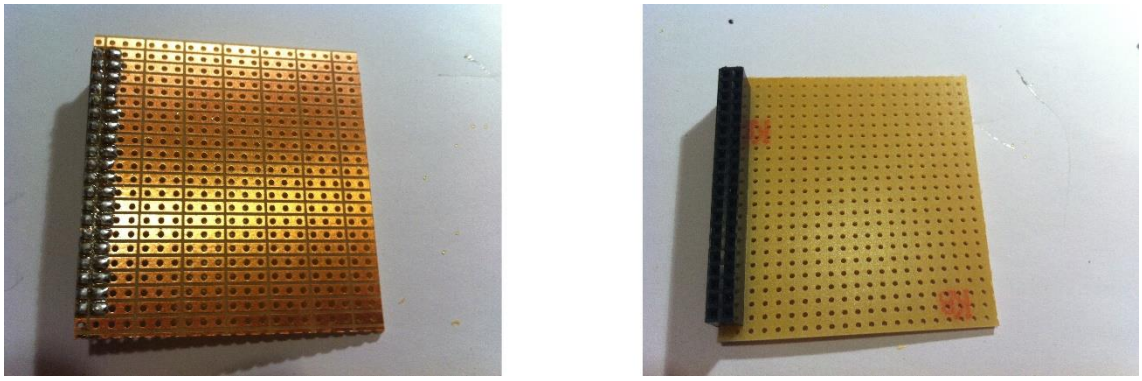

FIGURE 3. Connector on which hardware is soldered

Pieces of hardware, which connect to the GPIO are placed on a copper panel, and the panel is connected to the GPIO via a connector (see figure 3). This way it is not necessary to solder anything directly onto the RPi.

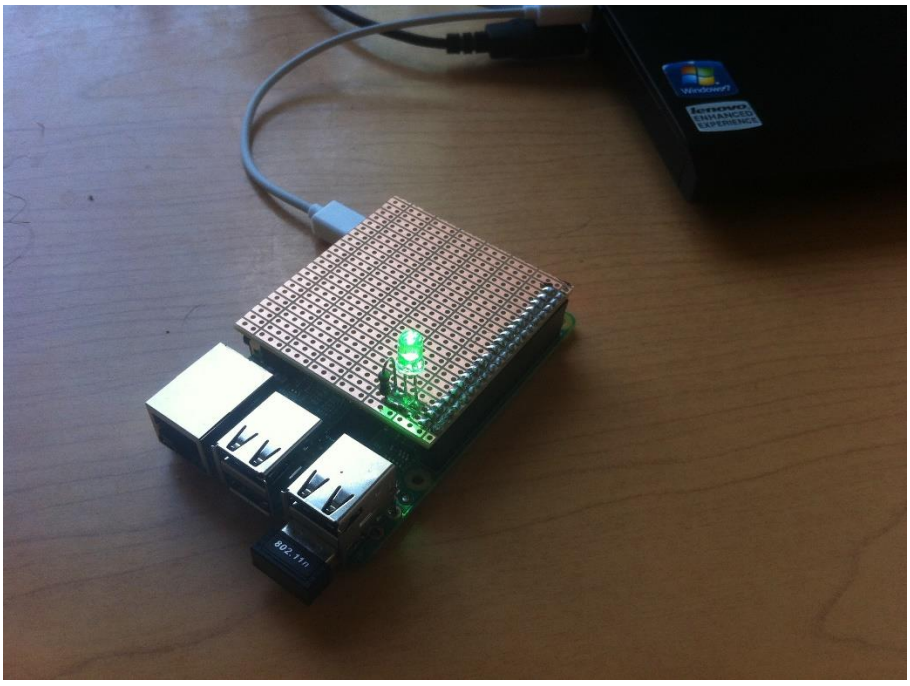

FIGURE 4. LED test

To test whether everything works as expected, a LED can be used (see figure 4).

3.4 DS18B20+ One Wire Digital Temperature Sensor

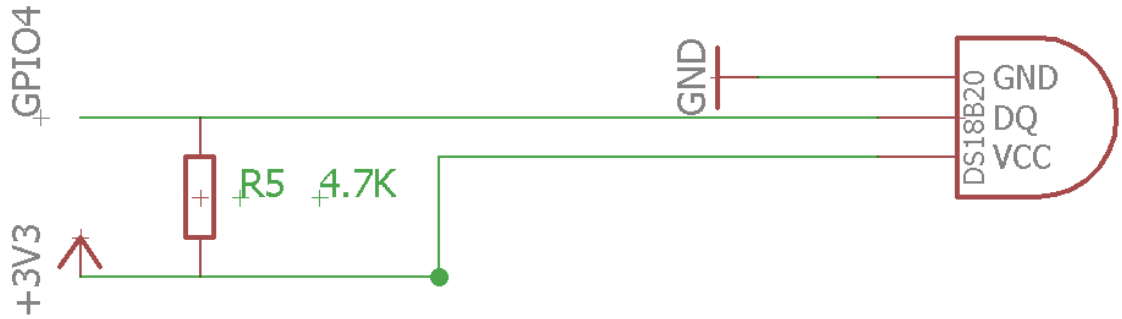

DIAGRAM 3. Thermometer

The temperature is measured using a one wire thermometer (see diagram 3) that is attached to the pins 1, 7 and 6. (5)

In 1-wire sensors all the data is sent using a single wire. This is very convenient on microcontrollers, such as the RPi. It is convenient, because it requires only one pin to transmit data. Most 1-wire sensors come with unique serial code, which means that it is possible to attach multiple instances of the same hardware without having them interfering. The 1-Wire technology was originally developed by Dallas Semiconductor Corp. The purpose is to send data and signaling at low speed while powering using one signal.

A network of 1-Wire devices with a master device is called a MicroLAN. Communication happens so that the master device (which can be a PC or a microcontroller) initiates communication. This helps avoiding collisions. In case of collision, the master device retries communication. All the devices of the same system are placed on one bus. Before the beginning of the communication, the master sends a “reset” signal, which will be received by all the devices. Then a select command is issued. Each device has a 64-bit identifier, and the 8 least significant bits hold the type of the device. The 8 most significant bits are for CRC purposes. (6)

3.5 HC-SR04 Ultrasound module

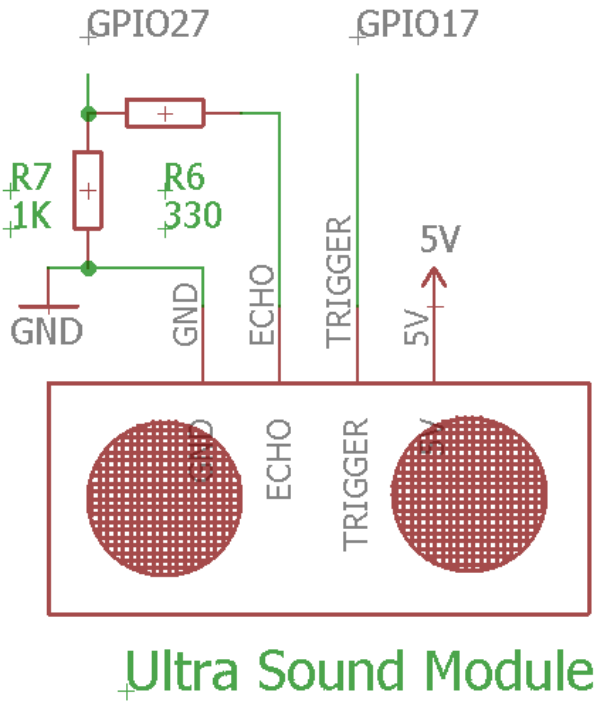

DIAGRAM 4. Ultrasound wave module

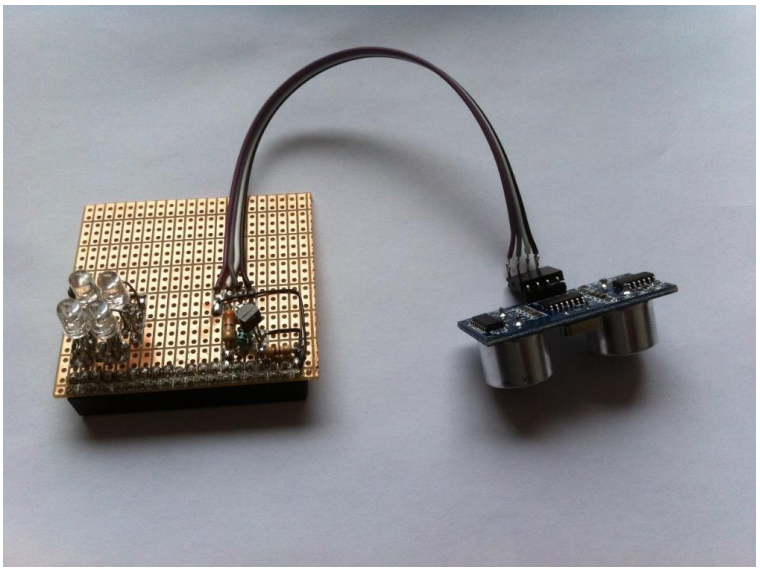

FIGURE 5. Ultrasound wave module soldered on a connector

The system is also equipped with an ultrasound module (see diagram 4 and figure 5). It is capable of sending out a soundwave and then it notifies when the wave echoes back. (7)

3.6 Camera module

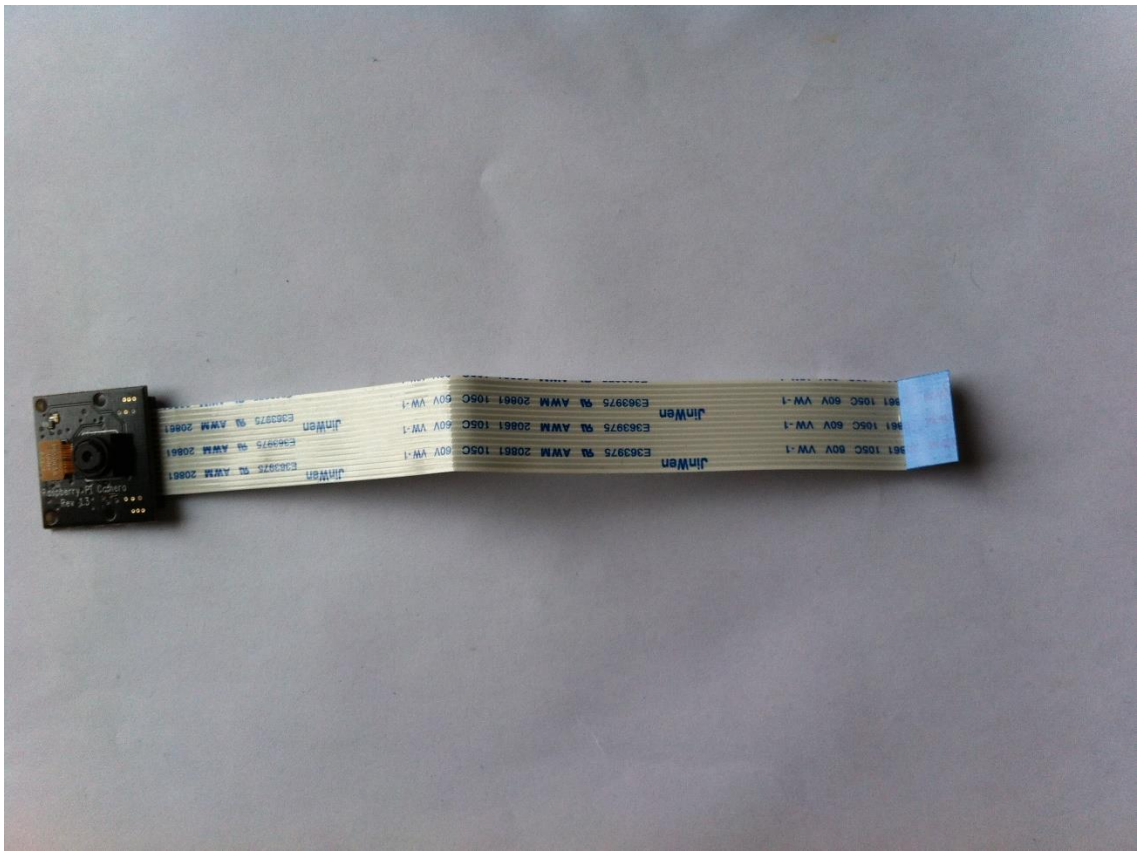

FIGURE 6. The Raspberry Pi camera module

In the Raspberry Pi camera module there is a camera attached to the CSI port (see figure 6). This camera can be used to capture high definition video as well as still images. There is also a standard application which provides built-in commands for capturing images. Raspistill and rastpstillyuv are for capturing still images, and raspivid is for video.

3.7 Testing with using LEDs

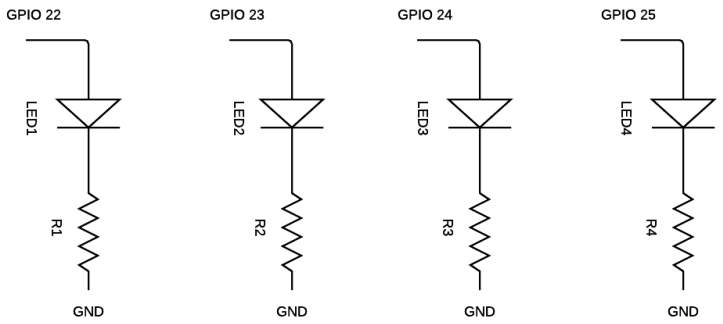

DIAGRAM 5. Testing with LEDs

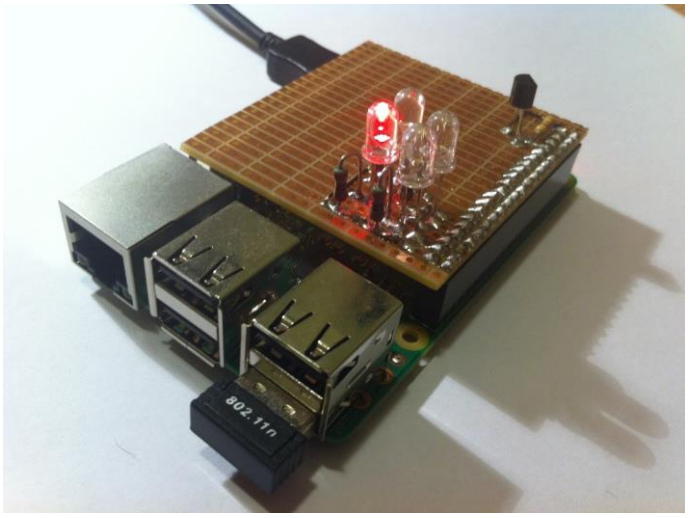

FIGURE 7. LEDs in action

Before the relays controlling the motors are installed, it is a good practice to test the software by replacing motors by simple LEDs (see diagram 5 and figure 7). These LEDs represent the electromotor; if there is HIGH put on a LED, it will light, which indicates that the motor is spinning. If the LEDs blink as expected, that means that the right GPIO ports will be written HIGH when the motor is requested to operate. Then it is allright to change the LEDs to relays and connect them to the batteries in the same time.

3.8 Relays

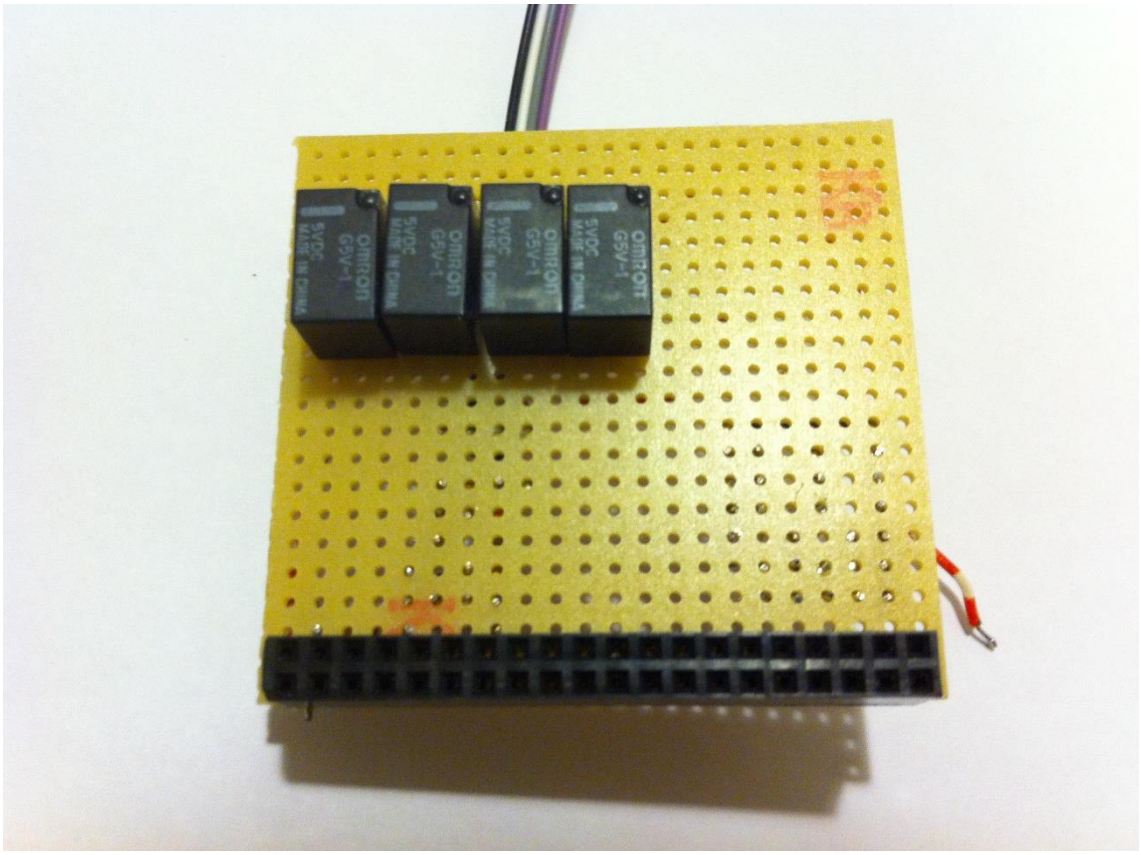

FIGURE 8. Relays on the connector board

It is not possible to put very high voltage through the CPU that operates the electromotors. Therefore, relays must be used. When some small voltage is put on the relay, it opens a connection between the 4xAA batteries of the car and the electromotor.

When current stops to flow through the coil in the relay, a little additional current might suddenly flow back to the opposite (bad) direction and it might damage the RPi. Therefore, it is recommended to make use of some relay protection. Diodes just provide a solution for this issue. Diodes allow current to flow only in one direction. Figure 8 shows as the above mentioned relays are installed onto a connection board.

3.9 Adafruit PowerBoost 500 Basic

FIGURE 9. The Adafruit PowerBoost 500 Basic

It is necessary to supply the RPi with power while the car is on the road. One option could be to install a separate battery that provides power. The other way, which is also chosen for this project, is to feed the RPi from the 4xAA batteries, which normally run the electromotors.

In that case, a stable 5V voltage is necessary. Operating the electromotors would temporarily drop the voltage of the batteries. In that case the RPi would suddenly power off every now and then.

The Adafruit PowerBoost module is just capable of providing a solution to this problem. This component stabilizes voltage coming from the battery. The batteries are connected to the PowerBoost and the PowerBoost is connected to the RPi to power it. Figure 9 shows the PowerBoost at its unpacked state.

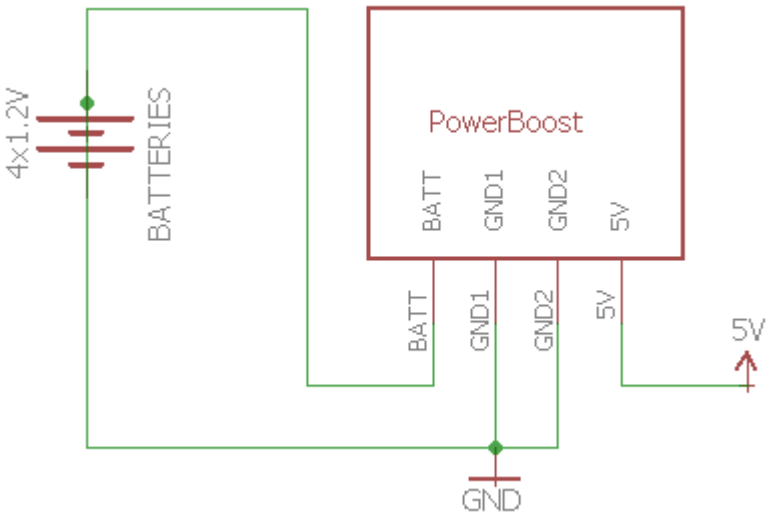

DIAGRAM 6. How to connect the Adafruit BowerBoost

Diagram 6, based on the official PowerBoost documentation (9) shows how the PowerBoost pins are connected to the batteries and the RPi.

There are a few more pins on the PowerBoost, namely:

- LB: This indicates when the battery voltage is lower than 3.2V. By default it is HIGH, and in case of a low voltage, it drops to LOW.
- EN: Enable pin. When this pin is connected to the ground, the PowerBooster turns off.

3.10 Overal circuit diagram

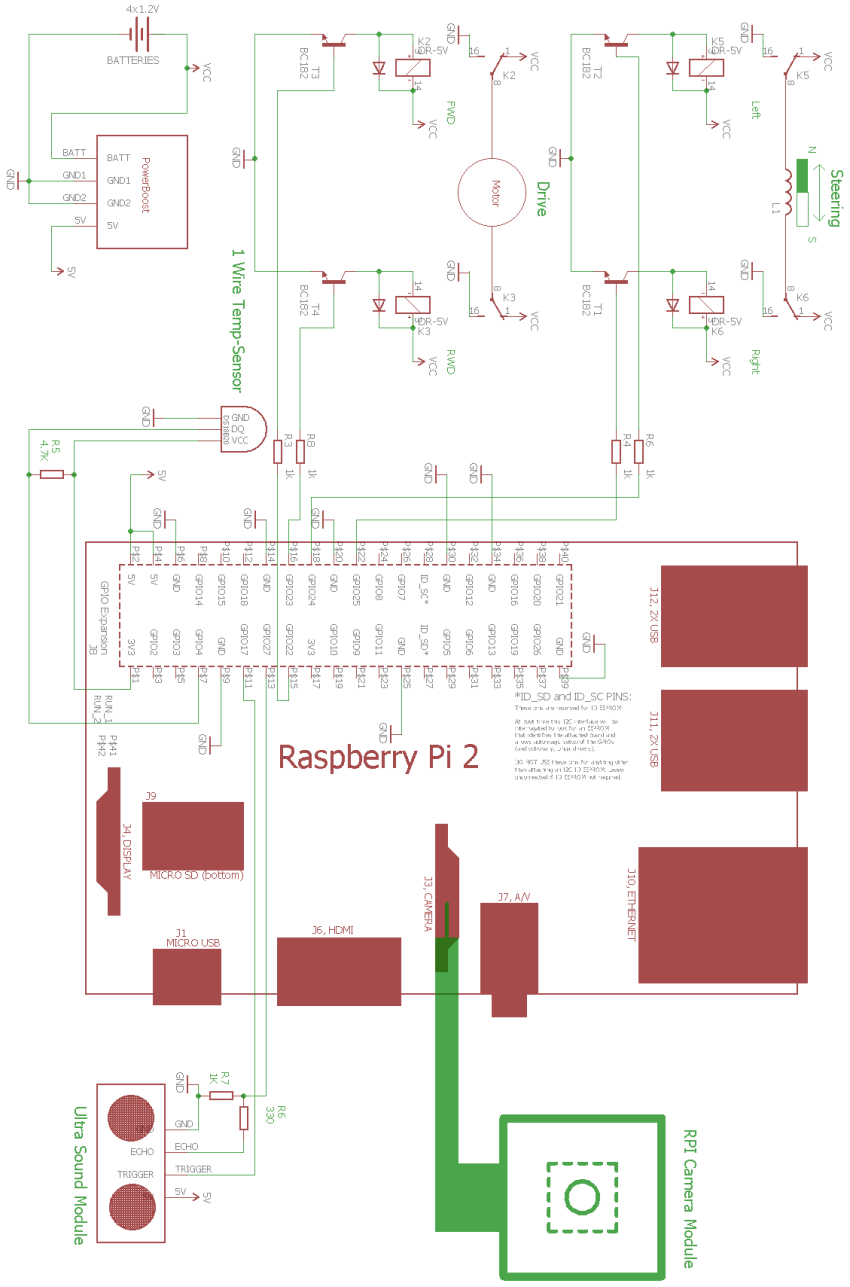

DIAGRAM 7. Overal circuit diagram

All the hardware components must be connected as shown on diagram 7.

In the diagram above, an external *Raspberry Pi 2* library was used. (10)

4 DEVELOPMENT ENVIRONMENT

4.1 For the server (RPi)

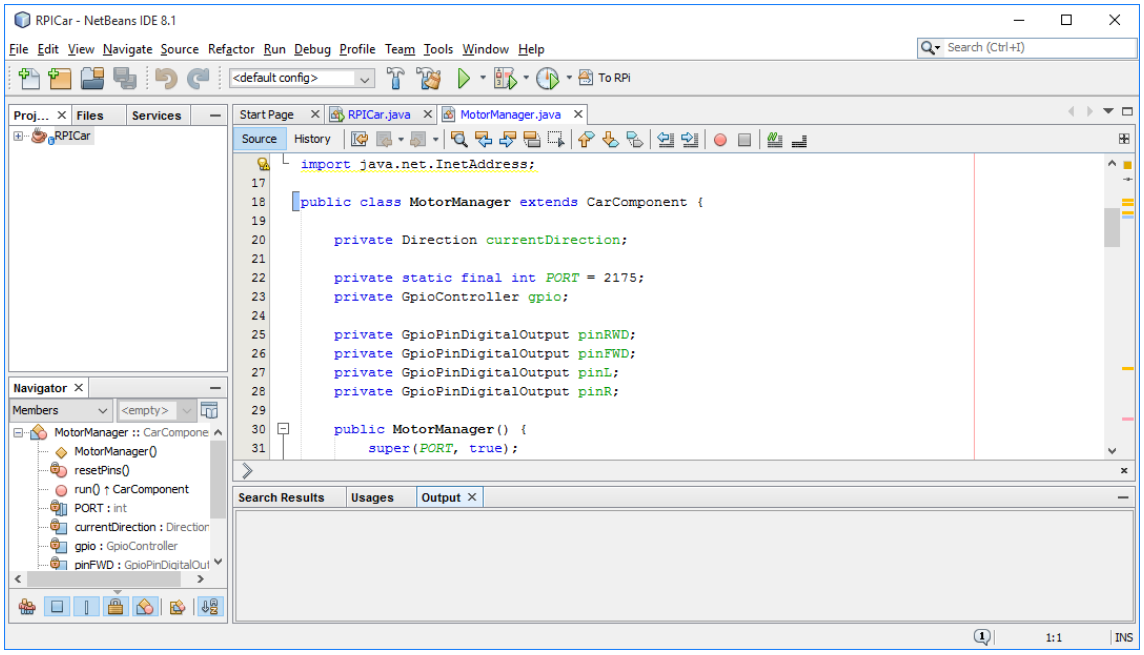

FIGURE 10. Netbeans

The software that runs on the RPi is written in Java. NetBeans provides a great API for fastening up the process of development using Ant scripts. Figure 10 shows NetBeans being used for developing the software for the client.

With Ant you can do such things as automatically call batch scripts and transfer binary code to the RPi.

```
<target name="scpcopy" depends="clean,package-for-store">
    <exec executable="cmd">
        <arg value="/c"/>
        <arg value="C:\Program Files\NetBeans 8.1\extide\ant\bin\SCPCopy.bat"/> <!-- BAT
        <arg value="pi"/> <!-- username -->
        <arg value="raspberry"/> <!-- pass -->
        <arg value="192.168.0.23"/> <!-- ip -->
        <arg value="C:\Users\tamas\Dropbox\code\RPICar\rpicar\RPICar\store\RPICar.jar"/>
    </exec>
</target>
```

DIAGRAM 19. Script that compiles, packs, and sends Java archive to RPi

Figure 9 shows a snapshot of an Ant script, which runs „clean, package-for-store”. Then executes SCPCopy.bat file with switches –pi –raspberry 192.168.0.23 and the path of the JAR file to be copied. As a result, the compiled program is sent to the RPi.

On the server side Java SE 8 and Netbeans 8.1 IDE was used.

4.2 For the client (Android platform)

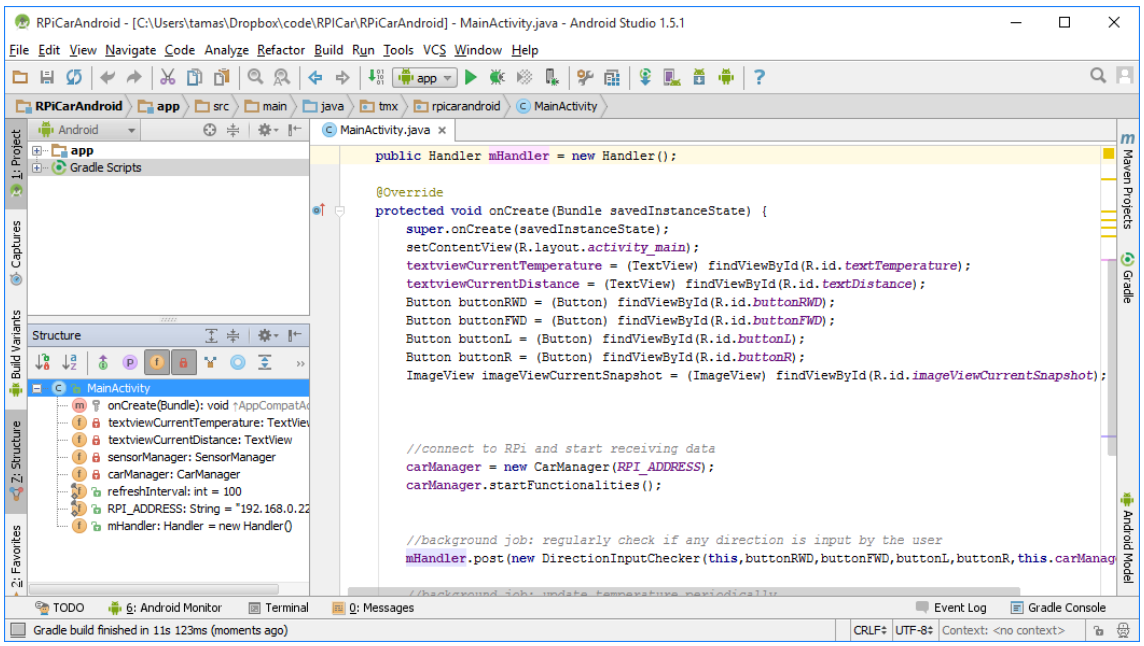

FIGURE 11. Android Studio

For the Android platform Google's Android studio is used. It's an easy-to-use IDE. Figure 11 shows Android studion being used for developing the client software.

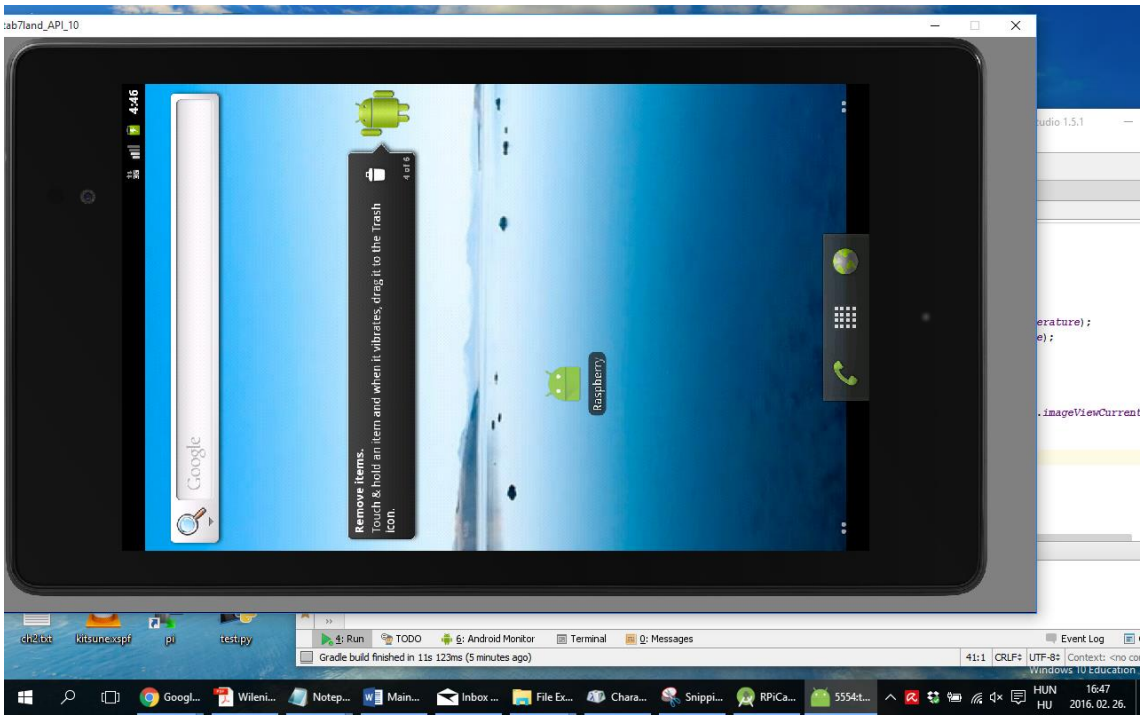

FIGURE 12. Emulating an Android device on PC

The tablet used for this thesis project runs Android 5 (Lollipop). The development environment was Android Studio 1.5.1.

The Android studio also supports device virtualization (AVD). Using AVD it is not necessary to run the program on a physical device, but it is possible to create a virtual device with a custom screen size and hardware parameters (see figure 12).

5 OVERVIEW OF THE SOFTWARE

Two pieces of software and one shared class library were developed altogether during this thesis project.

- A software for the server side. This software runs on the Raspberry Pi. It receives movement commands from the client and collects data and shoots images. The data and images collected are sent to the client.
- A software for the client side. This software runs on the Android tablet. This software connects to the sever, sends movement commands and receives environment data and images.
- There is also a library which contains classes and an interface that are shared by the client and the server.

The contents of the shared class library are the following:

- CarComponent (abstract class)
- Direction (class)
- EnvironmentData (class)

5.1 The server side (RPi)

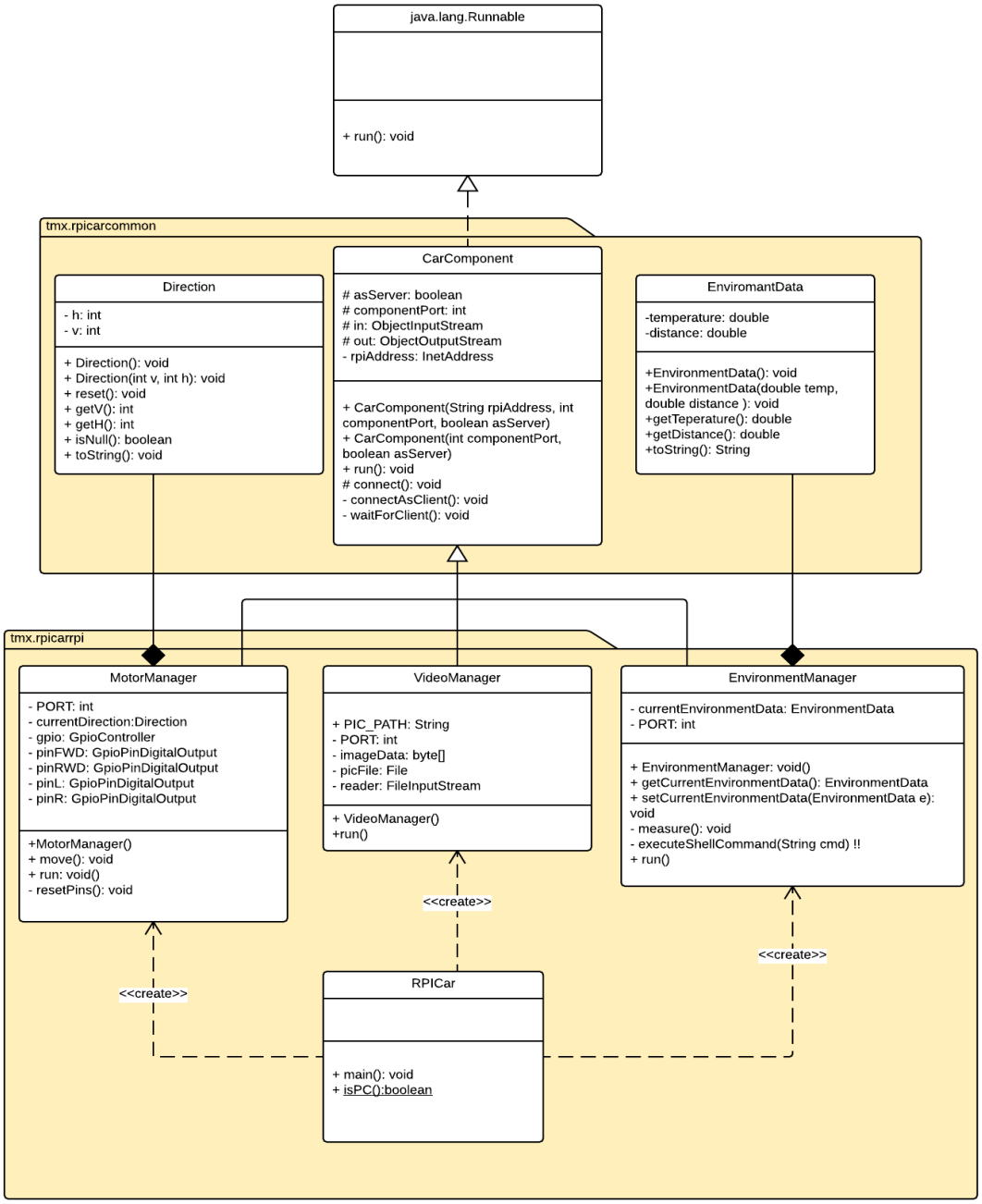

DIAGRAM 8. Server-side class diagram

The class diagram (diagram 8) above shows the software architecture implemented on the Rpi, which acts as the server.

There are three classes that implement the three main tasks of the RPi. They are the MotorManager, VideoManager, and the EnvironmentManager. They all inherit from CarComponent. They are all *Runnable*s and are responsible for exchanging data with the client.

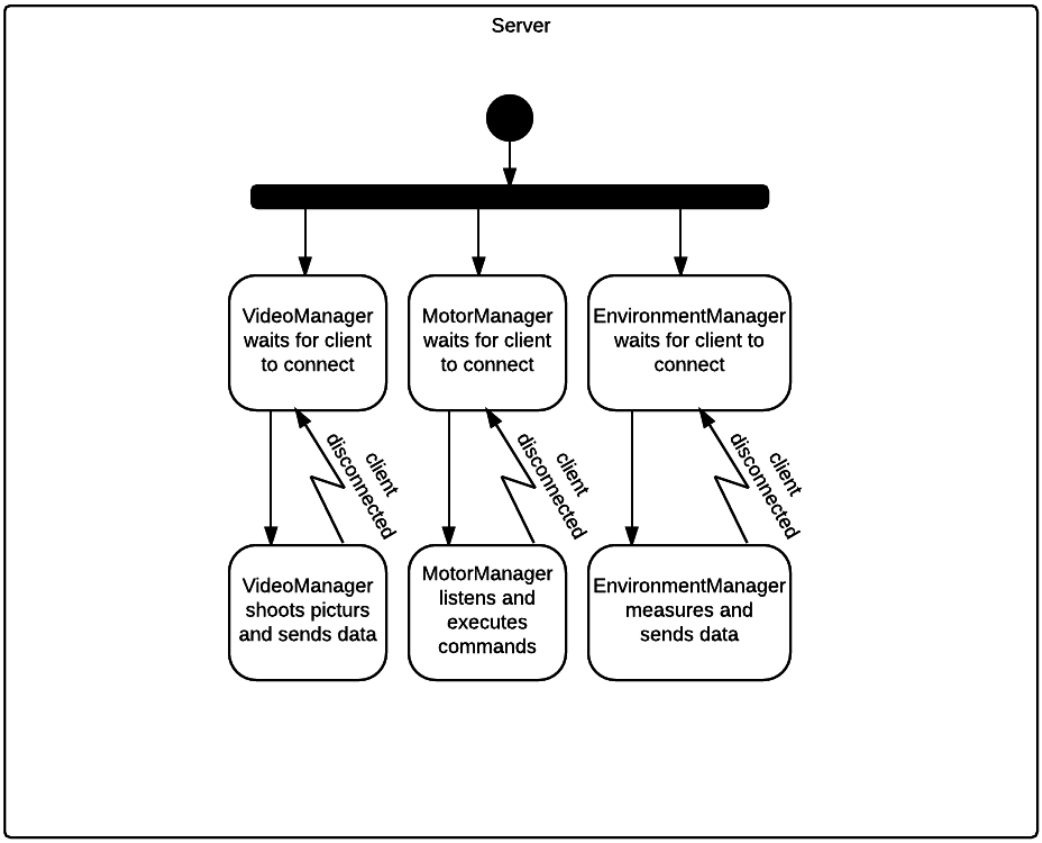

DIAGRAM 9. Server-side activity diagram

The above activity diagram (diagram) depicts how the system works from an abstract point of view. Upon startup, three threads get started in parallel. They all wait for a client to connect. As soon as the client connects, they start to transmit data and work. In the unlucky event that the client gets disconnected, they fall back to the earlier state and wait to be connected again.

5.1.1 CarComponent

CarComponent is the baseclass for all the classes that deals with tasks that are run on independent threads where there is also a data transfer present. This class provides a TCP connectivity either as a server or as a client. CarComponent implements *java.lang.Runnable*, which is required to make subclasses of CarComponent on a runnable thread.

5.1.2 MotorManager

The responsibility of MotorManager is to receive and process motor movement commands which are sent by the client. When the command is received, the motor pins on the GPIO are set accordingly. There is also a field currentDirection of type Direction. This holds the current motor command that has to be executed.

5.1.3 Direction

Direction is a simple class that contains the information about a motor command. It has two fields. The field „h” represents horizontal direction. In case it is -1, it means a left turn. If it is 0, then no turning is on demand. If it is 1, then a right turn is required.

The other field „v” (vertical) works similarly. -1 stands for reverse, 0 means not to go forward nor backwards while 1 means drive forward.

The combination of the two fields above builds together a direction.

5.1.4 VideoManager

The role of VideoManager is to capture pictures and send them to the cient. The image capturing relies on the standard app that is shipped by Raspberry. In order to capture an image, the following terminal command has to be issued:

```
raspistill -w 150 -h 150 -o currentSnapshot.jpg -t 1 –nopreview
```

This makes a picture, which is of resolution 150×150, and saves it into currentSnapshot.jpg. Afterwards the created image is sent to the client. The image is sent over by encoding the jpeg file using a base64 algorythm.

5.1.5 EnvironmentManager

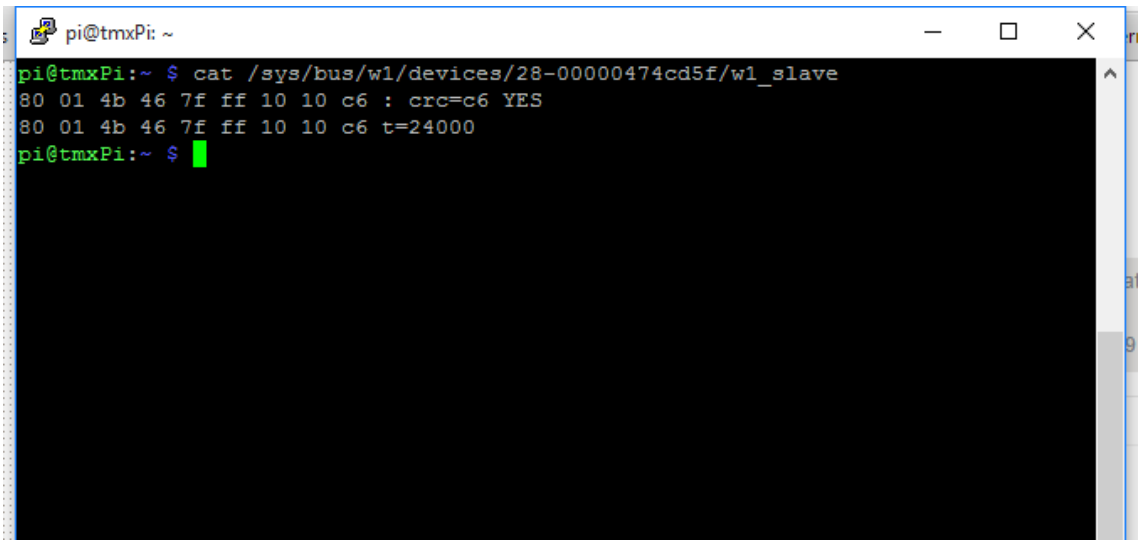

FIGURE 13. Result of measuring temperature with the thermometer

The EnvironmentManager class collects data from the environment. This includes temperature and distance ahead the next physical obstacle. The current value can be read out of the thermometer by issuing a shell command (see figure 13). More precisely a sysfs file has to be echoed out as follows:

```
cat /sys/bus/w1/devices/28-00000474cd5f/w1_slave
```

However, from here we only need whatever stands after t=…. So we use a regular expression to grab the necessary part of the chunk of text:

```
cat /sys/bus/w1/devices/28-00000474cd5f/w1_slave | grep -o

[0-9][0-9][0-9][0-9][0-9]
```

As a result, now we only have a bare number that indicates the current temperature in Celsius degrees multiplied by 1000.

Getting the distance measured by the ultrasonic module is a bit more complicated. To request a measurement on the ultrasonic module, its TRIGGER pin has to be written HIGH first. Once the module has received the signal, it writes HIGH onto its ECHO pin. This is the moment when it sends out the ultrasound wave. When the wave arrives back, it writes HIGH again to its ECHO pin. In this way we know how much time it took for the soundwave to reflect. From here only a simple calculation is needed to know the distance in cm:

$$distance = (elapsed \times 34300)/2$$

5.2 The client

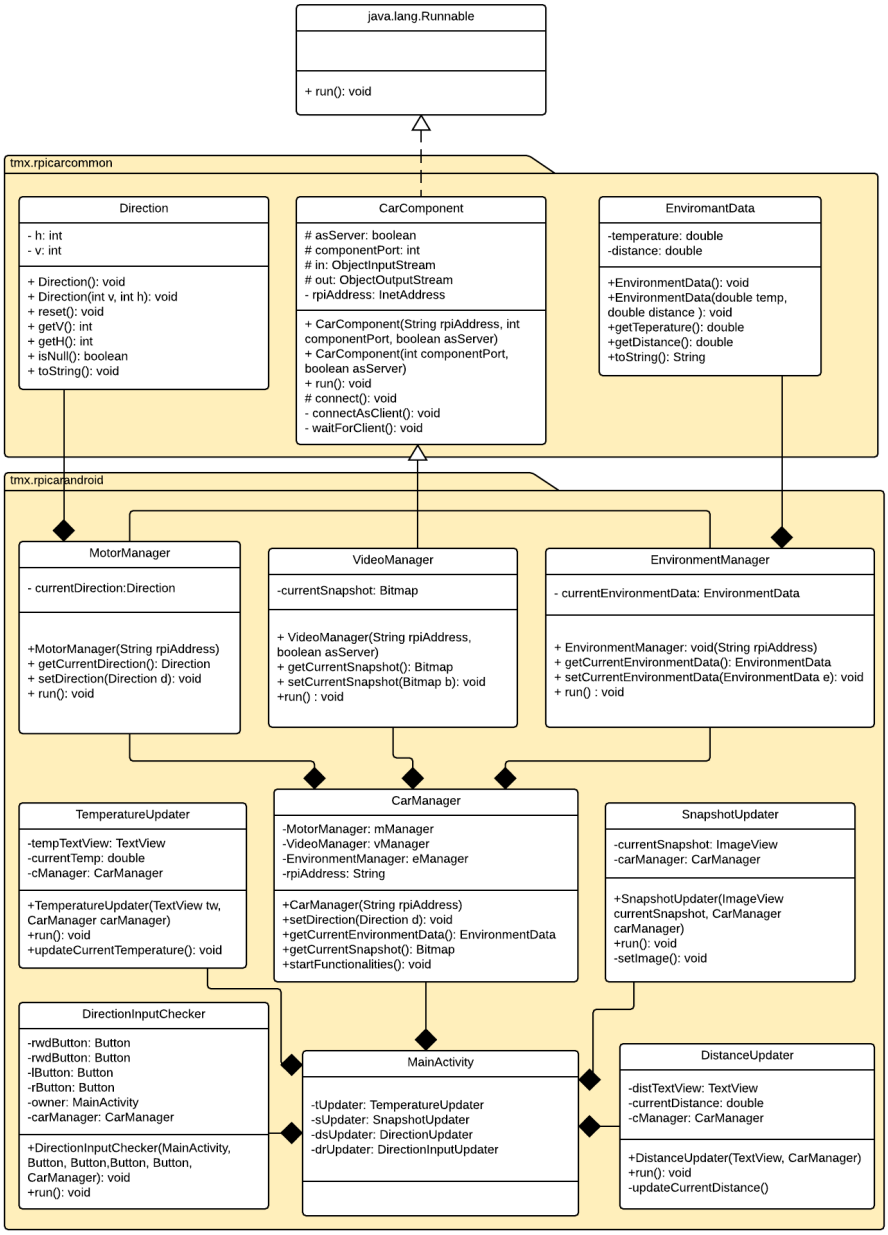

DIAGRAM 10. Client-side class diagram

The diagram 10 above visualuizes the architecture implemented on the client side. The classes found in the first part work similar to the ones on the server-side, except that this time the data sender and receiver roles are changed.

The MotorManager, VideoManager and EnvironmentManager classes are responsible for exchanging data with the server. They are managed by the CarManager. A reference to a CarManager instance can be found in MainActivity.

The DirectionInputChecker, TemperatureUpdater, SnapshotUpdater and DistanceUpdater are responsible for interacting with the graphical user interface. References to these class instances are contained directly in the MainActivity.

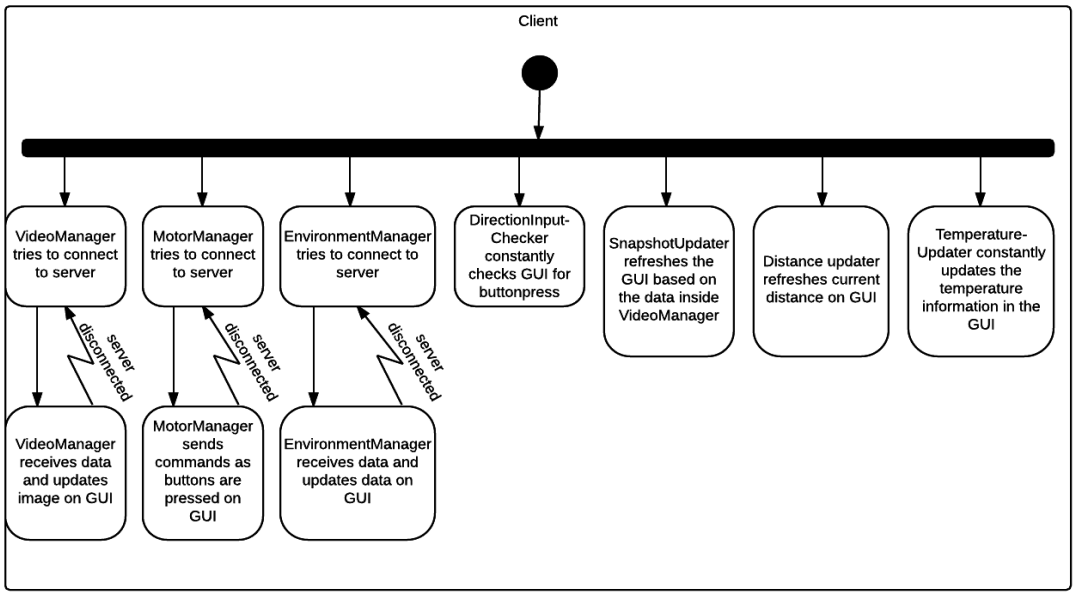

DIAGRAM 11.Client-side activity diagram

On the client side, the architecture is somewhat similar as on the Server. Here there are a few more classes that are responsible for interacting with the user interface (see diagram 11).

5.2.1. CarManager

The CarManager class provides an easy way to give commands and receive information from the classes VideoManager, MotorManager and EnvironmentManager. The CarManager basically contains references to the above mentioned classes and has Thread object references of them.

5.2.1 MotorManager

The task of MotorManager is to send the motor commands to the server based on which buttons are pressed on the GUI. The MotorManager only sends out motor commands if the current desired direction is not a „null-vector” (at least one direction button is pressed). MotorManager also owns an instance of Direction, (see section 3.1.3.).

5.2.2 VideoManager

VideoManager constantly receives images sent by the server and stores them in a Bitmap object „currentSnapshot”. This is publically readable through its function getCurrentSnapshot. Image data is received by decoding the jpeg image from the base64 algorithm sent by the server.

5.2.3 EnvironmentManager

EnvironmentManager constantly reads incoming EnvironmentData objects sent by the RPi. The received data is stored in its currentEnvironmentData field.

5.2.4 MainActivity

This class contains code that runs while showing the graphical user interface. It runs 4 background threads, which are discussed below.

5.2.5 DistanceUpdater

The DistanceUpdater class updates the text field of current distance ahead, based on what is the last data received from the RPi. Data is retrieved indirectly from CarManager and EnvironmentManager.

5.2.6 SnapshotUpdater

SnapshotUpdater works similarly to DistanceUpdater: it gets data from the VideoManager (through CarManager) and places the most recent image on the screen.

5.2.7 DirectionInputChecker

The DirectionInputChecker class constantly checks the four direction buttons on the screen. When one or several buttons get pressed on the screen, it immediately sends out those motor commands to the RPi through the CarManager.

5.2.8 TemperatureUpdater

Similarly to DistanceUpdater, TemperatureUpdater updates the TextView of the most recently measured temperature.

5.3 The pi4j Library

Handling the GPIOs is not possible with the standard Java API library.

Therefore, an external third-party library, *pi4j* has been used. (11)

An example on how to use this library:

```
// create gpio controller instance
final GpioController gpio = GpioFactory.getInstance();

// provision gpio pin #02 as an input pin with its internal pull down
//resistor enabled
// (configure pin edge to both rising and falling to get
//notified for HIGH and LOW state
// changes)
GpioPinDigitalInput myButton =
gpio.provisionDigitalInputPin(RaspiPin.GPIO_02, "MyButton",
PinPullResistance.PULL_DOWN); // PIN RESISTANCE (optional)
```

Now it is possible to control the pin.

```
// explicitly set a state on the pin object
myLed.setState(PinState.HIGH);

// use convenience wrapper method to set state on the pin object
myLed.low();
myLed.high();

// use toggle method to apply inverse state on the pin object
myLed.toggle();

// use pulse method to set the pin to the HIGH state for
// an explicit length of time in milliseconds
myLed.pulse(1000);
```

6 CONCLUSION AND POSSIBILITY OF FURTHER DEVELOPMENT

When I started this project, I had a quite brief idea how what would be implemented. The main main for me was to use and practice my skills in using those technologies involved.

I have learned a lot about working with Java and Android. I have also gained some experience in telecommunication. It is quite certain that I did not manage to do everything perfectly, I have still developed a lot. Fortunately, these mistakes were not deadly, but they rather had a constructive effect. All in all, I am proud of my work and all the efforts that I invested into it.

The key points where this project could be mainly improved:

1. The communication protocol between the RPi and the client platform could be more platform independent. At this point data is (de)serialized into Java objects. One nicer solution could http/soap messages.

2. The server prorgram language is Java. Java is a platform idependent language per se. However, not all microcontrollers support running a Java Runtime Environment. That being said it would not be possible to run the same code eg. on an Arduino for example.

3. Running the client application is also quite strictly bound to Android. It was written directly with Android Studio, which means that this code is hard to port into other platforms, such as Apple iOS, or Windows (Either with PC or Mobile). A solution to this could have been to implement the application using a platform independent developing environment. Xamarin could be a way to go.

4. Of course, the existing functionalities could be improved by more sensors and functionalities, such as an infrared lamp for night vision, or by security mechanisms, such as to automatically stop when an object is too close ahead, horn. Only fantasy defines the limit.

5. A better user interface is also optional, for example introducing a turn-by-tilting, joystick/wheel support, or even more, a virtual reality support. Although it would be quite an expensive hardware for a thesis project at this point, it would probably increase the user experience a lot.

6. The usage of such hardware that allows a smoother acceleration (a Pulse-width modulation).

7. Smoother image reception. At this point a video transition happens so that one picture is sent each second. It would be better to implement a real video stream.

8. It might also be a huge improvement to control the car via the Internet. In that way the car would not have to be required to be in the same network.

REFERENCES

1. Raspberry Pi Model B+ 2015. Raspberry Pi. Date of retrieval 19.04.2016 https://www.raspberrypi.org/products/raspberry-pi-2-model-b/.
2. Wikipedia 2016a. Raspberry Pi. Date of retrieval 21.04.2016 https://en.wikipedia.org/wiki/Raspberry_Pi
3. Wikipedia 2016b. Assembly Language. Date of retrieval 25.04.2016 https://en.wikipedia.org/wiki/Assembly_language
4. Wikipedia 2016c. Write once, run everywhere. Date of retrieval 30.04.2016 https://en.wikipedia.org/wiki/Write_once,_run_anywhere
5. DS18B20+ datasheet. Date of retrieval 2016.10.13 http://datasheets.maximintegrated.com/en/ds/DS18B20.pdf
6. Wikipedia 2016d. 1-Wire. Date of retrieval 10.05.2016 https://en.wikipedia.org/wiki/1-Wire
7. HC-SR04 Ultrasound Ranging Module datasheet. Date of Retrieval 2016.10.13 https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf
8. Adafruit PowerBoost 500 Basic pinout. Date of retrieval 13.10.2016 https://learn.adafruit.com/adafruit-powerboost/pinouts
9. Cadsoft Raspberry Pi 2 Library. Date of retrieval 06.10.2016 https://www.element14.com/community/docs/DOC-74714/l/the-raspberry-pi-2-cadsoft-library
10. pi4j library. Date of retrieval 01.02.2016 http://pi4j.com/